Department of Signal Processing and Acoustics

Morfessor 2.0: Python Implementation and Extensions for Morfessor Baseline

Sami Virpioja
Peter Smit
Stig-Arne Grönroos
Mikko Kurimo

Aalto University

SCIENCE + TECHNOLOGY

TECHNICAL REPORT

Aalto University publication series
SCIENCE + TECHNOLOGY 25/2013

Morfessor 2.0: Python Implementation and Extensions for Morfessor Baseline

Sami Virpioja
Peter Smit
Stig-Arne Grönroos
Mikko Kurimo

Aalto University
School of Electrical Engineering
Department of Signal Processing and Acoustics

Affiliations
This work was performed in two departments of Aalto University.

Department of Information and Computer Science, School of Science:
Sami Virpioja
Peter Smit (until December 2012)
Mikko Kurimo (until November 2012)

Department of Signal Processing and Acoustics, School of Electrical Engineering:
Peter Smit (January 2013 onwards)
Stig-Arne Grönroos
Mikko Kurimo (December 2012 onwards)

Aalto University publication series
SCIENCE + TECHNOLOGY 25/2013

© Sami Virpioja, Peter Smit, Stig-Arne Grönroos and Mikko Kurimo

ISBN 978-952-60-5501-5 (pdf)
ISSN-L 1799-4896
ISSN 1799-4896 (printed)
ISSN 1799-490X (pdf)
http://urn.fi/URN:ISBN:978-952-60-5501-5

Unigrafia Oy
Helsinki 2013

Finland

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n°287678 and the Academy of Finland under the Finnish Centre of Excellence Program 2012-2017 (grant n°251170) and the LASTU Programme (grant n°256887 and 259934). The experiments were performed using computer resources within the Aalto University School of Science "Science-IT" project.

A! **Aalto University**

Aalto University, P.O. Box 11000, FI-00076 Aalto www.aalto.fi

Abstract

Author
Sami Virpioja, Peter Smit, Stig-Arne Grönroos and Mikko Kurimo

Name of the publication
Morfessor 2.0: Python Implementation and Extensions for Morfessor Baseline

Publisher School of Electrical Engineering

Unit Department of Signal Processing and Acoustics

Series Aalto University publication series SCIENCE + TECHNOLOGY 25/2013

Field of research Computer and Information Science, Speech and Language processing

Abstract

Morfessor is a family of probabilistic machine learning methods that find morphological segmentations for words of a natural language, based solely on raw text data. After the release of the public implementations of the Morfessor Baseline and Categories-MAP methods in 2005, they have become popular as automatic tools for processing morphologically complex languages for applications such as speech recognition and machine translation. This report describes a new implementation of the Morfessor Baseline method. The new version not only fixes the main restrictions of the previous software, but also includes recent methodological extensions such as semi-supervised learning, which can make use of small amounts of manually segmented words. Experimental results for the various features of the implementation are reported for English and Finnish segmentation tasks.

Keywords morpheme segmentation, morphology induction, unsupervised learning, semi-supervised learning, morfessor, machine learning

ISBN (printed) **ISBN (pdf)** 978-952-60-5501-5

ISSN-L 1799-4896 **ISSN (printed)** 1799-4896 **ISSN (pdf)** 1799-490X

Location of publisher Helsinki **Location of printing** Helsinki **Year** 2013

Pages 32 **urn** http://urn.fi/URN:ISBN:978-952-60-5501-5

Contents

1 Introduction 2
  1.1 Terminology 3
  1.2 Workflow 4
2 Method 4
  2.1 Model and Cost Function 5
  2.2 Training and Decoding Algorithms 8
3 Evaluation 18
  3.1 Boundary Precision and Recall Evaluation 18
  3.2 Statistical Significance Testing 19
4 Experiments 19
  4.1 Comparison to Morfessor 1.0 20
  4.2 Random Variation in Model Training 23
  4.3 Random Split Initialization 23
  4.4 Training Speed-up with Skips 23
  4.5 Viterbi Training 24
  4.6 On-line Training 25
  4.7 Frequency Dampening 26
  4.8 Weight Optimization 27
  4.9 Semi-supervised Training 27
5 Conclusions 29

1 Introduction

Morfessor, originally developed by Creutz and Lagus (2002, 2004, 2005b,a, 2007), is a family of methods for unsupervised learning of morphological segmentation. It is mostly targeted to languages with complex but concatenative morphology, such as Finnish and Turkish, but is useful for any language with compound words or non-fusional inflections. It has been widely used in natural language processing applications such as speech recognition (e.g. Hirsimäki et al., 2006; Creutz et al., 2007; Mihajlik et al., 2010; Gelas et al., 2012), speech retrieval (e.g. Arisoy et al., 2009; Turunen and Kurimo, 2011), and statistical machine translation (e.g. Virpioja et al., 2007; Luong et al., 2010; Mermer and Akın, 2010; Clifton and Sarkar, 2011; Popović, 2011).

The most popular versions of Morfessor are Morfessor Baseline (Creutz and Lagus, 2002, 2005b) and Morfessor Categories-MAP (Creutz and Lagus, 2005a, 2007). Both are based on probabilistic generative models that use sparse priors inspired by the Minimum Description Length (MDL) principle by Rissanen (1978). One reason for their popularity are the Perl implementations released under the GNU General Public Licence in 2005. Somewhat outdated by now, the implementations do not support Unicode data, lack library interfaces, and naturally do not include any of the more recent developments of method.

In this report, we present a new implementation of the Morfessor Baseline method, called *Morfessor 2.0*. It is meant to replace the old Morfessor 1.0 software (Creutz and Lagus, 2005b). The new features of Morfessor 2.0 compared to Morfessor 1.0 are the following:

- Design
  - Python source code compatible with Python 2.7, 3.2+ and PyPy
  - Both command line and library interfaces
  - Modular, easily extensible code
  - Full Unicode support
  - Direct support for related segmentation tasks such as chunking
  - Possibility to train the model directly from corpus
- Algorithms

– Random split initialization for batch training

– On-line training

– Training speed-up with random skips

– Frequency threshold and dampening for words in training data (Virpioja et al., 2011a)

– Semi-supervised learning from annotated training set (Kohonen et al., 2010a)

– Possibility to weight training data likelihoods (Kohonen et al., 2010a; Virpioja et al., 2011a)

– Optimization of data likelihood weight based on development set

– Viterbi training

– Forward algorithm and n-best Viterbi decoding

– Integrated boundary precision and recall evaluation and statistical significance testing with the Wilcoxon signed-rank test

1.1 Terminology

Morfessor 2.0 has been designed to be indifferent to the segmentation task at hand. Thus, to describe the task, we will use general terms that are not specific to the problem of morphological segmentation.

First, the smallest pieces of text that the algorithm processes are called *atoms*. The input for the learning algorithm is a set of sequences of atoms called *compounds*. The task of the algorithm is to find lexical units that are something between atoms and compounds; they are called *constructions*. Starting from the longest unit: compounds are sequences of constructions, which are sequences of atoms. Note that the sequences can contain only one item: for example, a compound can consist of only a single atom despite its name. Table 1 shows how these terms relate to three common segmentation tasks: morphological segmentation, word segmentation, and shallow parsing (chunking).

The operation of replacing a compound with its constructions is called *tokenization* and the reverse operation is called *detokenization*. Tokenization is

Table 1. Examples of segmentation tasks and terminology.

Task	Compounds	Constructions	Atoms
Morphological segmentation	word forms	morphs	letters
Chinese word segmentation	sentences	words	letters
Chunking / shallow parsing	sentences	phrases	words

performed by a *decoding algorithm*, that tries to find the most likely constructions for the given compound. Finally, the set of constructions that the algorithm finds from the input data is called simply a *lexicon*.

1.2 Workflow

As typical for machine learning methods, there are two phases when using Morfessor. We call them *training* and *decoding*. Input for the training phase is a set of compounds $\boldsymbol{D}_W$ and optionally a set of annotated compounds $\boldsymbol{D}_{W \to A}$, and the output are the model parameters $\boldsymbol{\theta}$. The performance of the method is evaluated on the decoding (tokenization) results A for a test data set, which is separate and independent from the training data sets. This workflow is illustrated by Figure 1. The cost function $L(\cdot)$ and tokenization function $\phi(\cdot)$ are described more thoroughly in the next section.

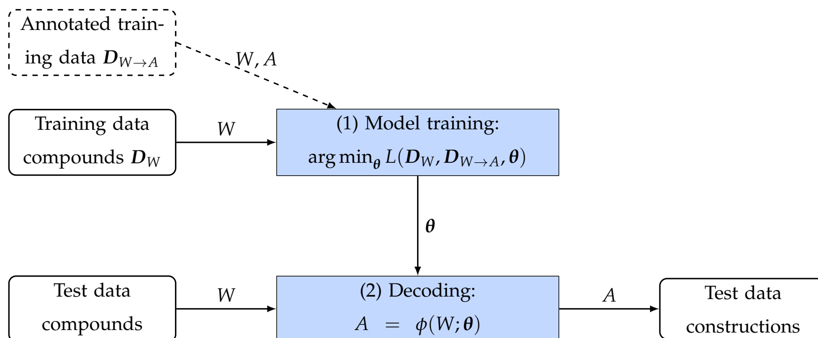


Figure 1. The standard workflow for Morfessor: The model is trained by selecting the model parameters $\boldsymbol{\theta}$ that minimize the cost function $L(\boldsymbol{D}_W, \boldsymbol{D}_{W \to A}, \boldsymbol{\theta})$ with compounds W in unannotated training data $\boldsymbol{D}_W$, and optionally compounds W and their constructions A in annotated training data $\boldsymbol{D}_{W \to A}$. The parameters of the trained model are used to tokenize the new compounds in test data.

2 Method

Each version of Morfessor can be characterized by three components: model, cost function, and training and decoding algorithms. In this section, we describe the components for the original Morfessor Baseline (Creutz and Lagus,

2005b) and introduce the changes implemented in the current version. The mathematical notation used in this report is based on Kohonen et al. (2010a), Virpioja et al. (2011a), and Virpioja (2012), and it is slightly different from the one used in the original Morfessor articles. We will mostly follow the presentation by Virpioja (2012, Section 6.4.1).

All Morfessor models consist of two parts: a lexicon and a grammar. The lexicon stores the properties of the constructions and the grammar determines how the constructions can be combined to form compounds. The grammar of Morfessor Baseline has two basic assumptions. The first is that a compound consists of one or more constructions. The only upper limit for the number of constructions in a compound is the number of atoms in the compound. In morphological segmentation, this is important especially for languages with many morphemes per word. The second assumption is that the constructions of a compound occur independently. While this is evidently a false assumption—and probably the most serious drawback of the method—it enables very efficient training algorithms.

The cost function of Morfessor Baseline is derived from the *maximum a posteriori* (MAP) estimate for the model. Thus it consists two parts, model likelihood and prior. The likelihood is derived directly from the model assumptions above. The prior, which determines the probability of the model lexicon, draws inspiration from the Minimum Description Length (MDL) principle: it is non-informative and based on efficient compression schemes.

The training algorithm of Morfessor Baseline can be characterized as greedy and local search. It starts with an initial lexicon, which usually consists of all the compound forms seen in the training data. Then it selects one compound at a time and tries simultaneously find the optimal segmentation for the compound and the optimal lexicon given the new segmentation and the segmentations of all the other known compounds. The training is normally done as a batch job; however, the new implementation supports also on-line training.

For decoding—finding the optimal segmentations for new compound forms without changing the model parameters—Morfessor Baseline applies a variation of the Viterbi algorithm. The Viterbi algorithm can be used for training, too, although it has a few problems that hinder its usefulness.

2.1 Model and Cost Function

Formally, the models of the Morfessor family are generative probabilistic models that predict compounds W and their analyses A given model param-

eters $\boldsymbol{\theta}$. That is, they define the joint probability distribution $p(A, W \mid \boldsymbol{\theta})$. The analysis $\boldsymbol{a}$ of a compound $\boldsymbol{w}$ is specified by the tokenization function

$$\boldsymbol{a} = \phi(\boldsymbol{w}; \boldsymbol{\theta}) \tag{1}$$

In the case of segmentation, an analysis $\boldsymbol{a}$ is a list of non-overlapping segments (constructions) of the compounds: $\boldsymbol{a} = (m_1, \ldots, m_n)$. A compound can then be generated from an analysis by concatenating the segments: $\phi^{-1}(\boldsymbol{a}) = m_1 \ldots m_n = \boldsymbol{w}$. As the detokenization operation $\phi^{-1}(\cdot)$ is simply concatenation, it does not depend on the model parameters.

The cost function of Morfessor Baseline is derived using maximum a posteriori estimation. That is, the goal is to find the most likely parameters $\boldsymbol{\theta}$ given the observed training data $\boldsymbol{D}_W$:

$$\boldsymbol{\theta}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta} \mid \boldsymbol{D}_W) = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}) p(\boldsymbol{D}_W \mid \boldsymbol{\theta}) \tag{2}$$

Thus we are maximizing the product of the model prior $p(\boldsymbol{\theta})$ and the data likelihood $p(\boldsymbol{D}_W \mid \boldsymbol{\theta})$. As usual, the cost function to minimize is set as the minus logarithm of the product:

$$L(\boldsymbol{\theta}, \boldsymbol{D}_W) = -\log p(\boldsymbol{\theta}) - \log p(\boldsymbol{D}_W \mid \boldsymbol{\theta}). \tag{3}$$

Data likelihood

Let $\boldsymbol{D}_W$ be the training data that includes N compounds and the boundaries ($\#_w$) between them. Assuming that the probabilities of the compounds are independent, the log-likelihood of the data is

$$\begin{aligned} \log p(\boldsymbol{D}_W \mid \boldsymbol{\theta}) &= \sum_{j=1}^{N} \log p(W = \boldsymbol{w}_j \mid \boldsymbol{\theta}) \\ &= \sum_{j=1}^{N} \log \sum_{\boldsymbol{a} \in \Phi(\boldsymbol{w}_j)} p(A = \boldsymbol{a} \mid \boldsymbol{\theta}), \end{aligned} \tag{4}$$

where $\Phi(\boldsymbol{w}) = \{\boldsymbol{a} : \phi^{-1}(\boldsymbol{a}) = \boldsymbol{w}\}$. Instead of calculating the logarithm of the sum over all possible analyses for each word, a hidden variable Y is introduced. It assigns each compound $\boldsymbol{w}_j$ in the training data to single analysis in $\Phi(\boldsymbol{w}_j)$. Given the analyses for all words in the data, $\mathbf{Y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N)$,

$$\log p(\boldsymbol{D}_W \mid \boldsymbol{\theta}, \mathbf{Y}) = \sum_{j=1}^{N} \log p(y_j \mid \boldsymbol{\theta}) = \sum_{j=1}^{N} \log p(m_{j1}, \ldots, m_{j|\boldsymbol{y}_j|}, \#_w \mid \boldsymbol{\theta}) \tag{5}$$

where m_{ji} is the ith construction of the jth compound. As Morfessor Baseline assumes that the constructions occur independently, this simplifies to

$$\log p(\boldsymbol{D}_W \mid \boldsymbol{\theta}, \mathbf{Y}) = \sum_{j=1}^{N} \Big(\log p(\#_w \mid \boldsymbol{\theta}) + \sum_{i=1}^{|\boldsymbol{y}_j|} \log p(m_{ji} \mid \boldsymbol{\theta}) \Big). \tag{6}$$

Changes from Morfessor 1.0. Morfessor 1.0 neglects the need to encode the compound boundaries and calculates the log-likelihood directly as the sum of construction log-probabilities $\log p(m_{ji} \mid \boldsymbol{\theta})$.

Prior

In the general formulation of Morfessor (Creutz and Lagus, 2007), the model parameters $\boldsymbol{\theta}$ are divided into a lexicon $\mathcal{L}$ and grammar $\mathcal{G}$. The lexicon includes the properties of the constructions, while the grammar determines how the constructions can be combined to form compounds. Morfessor Baseline has no grammar parameters, so that part of the prior is omitted and $p(\boldsymbol{\theta}) = p(\mathcal{L})$.

The prior in Morfessor assigns higher probabilities to lexicons that store fewer and shorter constructions. The construction m_i is considered to be stored if $p(m_i \mid \boldsymbol{\theta}) > 0$. The probability of the lexicon of μ constructions is

$$p(\mathcal{L}) = p(\mu) \times p(\text{properties}(m_1), \ldots, \text{properties}(m_\mu)) \times \mu!. \qquad (7)$$

The factorial term is explained by the fact that there are $\mu!$ possible ways to order a set of μ items and the lexicon is equivalent for different orders of the same set of constructions. The prior for the lexicon size μ has negligible effect and is omitted. The properties of the constructions in Equation 7 are further divided into those related to *form* and *usage*.

In Morfessor Baseline, the form of a construction is simply the atoms that it is composed of. The forms are assumed to be independent. The probability of the form of the construction m_i is based on length distribution $p(L)$ and categorical distribution $p(C)$ of the sequence of atoms $\boldsymbol{\sigma}_i$:

$$p(\boldsymbol{\sigma}_i) = p(L = |\boldsymbol{\sigma}_i|) \prod_{j=1}^{|\sigma_i|} p(C = \boldsymbol{\sigma}_{ij}) \qquad (8)$$

An implicit exponential length prior is obtained by removing $p(L)$ and using an end-of-construction marker $\#_c$ as an additional atom in $p(C)$ (Creutz and Lagus, 2005b).

The usage properties include only the counts of the constructions $\tau_i \in \{1, \ldots \nu\}$, where $\nu = \sum_i \tau_i$ is the total token count of the constructions. The counts provide the maximum-likelihood estimates for the probabilities of the constructions: $p(m_i \mid \boldsymbol{\theta}) = \tau_i / (N + \nu)$, where N is the number of compound boundaries in the training data. Given μ and ν, a non-informative prior for the construction counts is

$$p(\tau_1, \ldots, \tau_\mu \mid \mu, \nu) = 1 / \binom{\nu - 1}{\mu - 1}. \qquad (9)$$

This gives an equal probability to each possible combination of τ_is. The prior for ν is omitted for its negligible effect.

Changes from Morfessor 1.0. Morfessor 2.0 supports only the implicit exponential length prior for the constructions. In Morfessor 1.0, the distribution of atoms $P(C)$ was estimated from the training data $\boldsymbol{D}_W$, while in Morfessor 2.0, it is determined based on their occurrences in the current lexicon. This makes the coding of the lexicon more optimal. However, also the numbers of occurrences have to be encoded. A non-informative prior equivalent to that of the construction counts in Equation 9 is applied. Priors for the numbers of atoms types and tokens is neglected, but a factorial term $n!$ for different permutations of the n atom types is included (cf. Equation 7).

2.2 Training and Decoding Algorithms

Next, we will provide an overview of the training and decoding algorithms for Morfessor Baseline. Decoding algorithms are used in tokenization to select the most likely analysis for the given compound. This task is simpler than training, because the model parameters are fixed.

There are two types of training schemes that we considered here: In *batch training*, a fixed training data set is completely processed in one iteration of the algorithm. In *on-line training*, the training data set is not known beforehand, and new samples of compounds are processed one at a time. Regarding the search algorithms for model parameters, we make a division to *local* algorithms, that make small changes based on a single compound at a time, and *global* algorithms, that update parameters based on estimates over the full data set. We also consider the question of hyperparameter selection and describe the semi-supervised training technique by Kohonen et al. (2010a).

The implemented unsupervised training algorithms for Morfessor Baseline are listed in Table 2. In addition to the standard local recursive baseline algorithm of Morfessor and the inherent training with global Viterbi segmentation, Morfessor 2.0 implements a local Viterbi training algorithm.

Table 2. Training algorithms for Morfessor Baseline supported by the current implementations.

Search algorithm	Morfessor 1.0	Morfessor 2.0
Recursive baseline	batch	batch / on-line
Viterbi, global	(batch)	(batch)
Viterbi, local	–	batch / on-line

Outline for Batch and On-Line Training Schemes

Before going into the specific training algorithms, we shortly show the outline of the supported training schemes as pseudocode with functional programming style: batch training with a global search algorithm (Algorithm 1), batch training with a local search algorithm (Algorithm 2), and on-line training with a local search algorithm (Algorithm 3).[1]

In batch training, ϵ is a threshold parameter for the convergence of the model parameters. In the implementation, it is set as a fraction (default 0.005) of the number of compounds in the training data. The default initialization (INITMODEL) for the model parameters and compound segmentations is to set each compound as one construction. With a local search algorithm, the compounds in the training data are processed in a random order (RANDOMPERMUTATION).

Algorithm 1 Batch training with a global algorithm.

```
function GLOBALBATCHTRAIN(D_W, ε)
    θ, Y ← INITMODEL(D_W)
    L_old ← ∞
    L_new ← L(D_W, θ, Y)
    while L_new < L_old − ε do
        θ, Y ← GLOBALSEARCH(D_W, θ, Y)
        L_old ← L_new
        L_new ← L(D_W, θ, Y)
    end while
    return θ, Y
end function
```

[1] While on-line training with a global search algorithm is possible in theory, it is not very practical: all the data samples seen so far would have to be processed after each new sample.

Algorithm 2 Batch training with a local algorithm.

```
function LOCALBATCHTRAIN(D_W, ε)
    θ, Y ← INITMODEL(D_W)
    L_old ← ∞
    L_new ← L(D_W, θ, Y)
    while L_new < L_old − ε do
        J ← RANDOMPERMUTATION(1, ..., N)
        for j ∈ J do
            θ, Y ← LOCALSEARCH(w_j, D_W, θ, Y)
        end for
        L_old ← L_new
        L_new ← L(D_W, θ, Y)
    end while
    return θ, Y
end function
```

The on-line training (Algorithm 3) stops when all the training samples in $\boldsymbol{D}_W$ are processed once. The already seen data samples ($\boldsymbol{D}$), their segmentations, and the model parameters are by default initialized as empty sets or sequences.

Algorithm 3 On-line training with a local algorithm.

```
function ONLINETRAIN(D_W, D = (), θ = (), Y = [])
    while w ∈ D_W do
        Append w to D
        θ, Y ← LOCALSEARCH(w, D, θ, Y)
    end while
    return θ, Y
end function
```

Parameter Initialization

As mentioned above, the default initialization for the model parameters in Morfessor Baseline is to include all the compounds in the training data to the lexicon. This can be considered as a conservative initialization, and it may not be the best option in all cases. In Morfessor 2.0, we have included a possibility to initialize the lexicon with shorter, randomly chosen constructions. All compounds in the training data are split so that for each possible break point (i.e., between every atom of every compound), the compound is split with given probability p_{split}, and the resulting constructions are collected to the lexicon (and $\mathbf{Y}$ is initialized correspondingly). If the probability is zero, the initial constructions will be compounds, and if it is one, they will be atoms.

Forward-backward Algorithm

Theoretically appealing global training algorithm for the Morfessor Baseline model would be the forward-backward algorithm for hidden Markov models (Baum, 1972). It is a special case of the expectation-maximization (EM) algorithm. Given the old parameters $\boldsymbol{\theta}^{(t-1)}$, a new estimate of the parameters are obtained by:

$$\begin{aligned}\boldsymbol{\theta}^{(t)} &= \arg\min_{\boldsymbol{\theta}} E_Y\big[L(\boldsymbol{\theta}, \boldsymbol{D}_W, Y) \mid \boldsymbol{D}_W, \boldsymbol{\theta}^{(t-1)}\big] \\ &= \arg\min_{\boldsymbol{\theta}} \sum_{\mathbf{Y}} p(\mathbf{Y} \mid \boldsymbol{D}_W, \boldsymbol{\theta}^{(t-1)}) L(\boldsymbol{\theta}, \boldsymbol{D}_W, \mathbf{Y}),\end{aligned} \tag{10}$$

where $\mathbf{Y}$ gives the assignments of the compounds $\boldsymbol{w}$ in the training data to their possible segmentations, and the MAP cost function is

$$L(\boldsymbol{\theta}, \boldsymbol{D}_W, \mathbf{Y}) = -\log p(\boldsymbol{\theta}) - \log p(\boldsymbol{D}_W \mid \mathbf{Y}, \boldsymbol{\theta}). \tag{11}$$

However, there are two problems in this approach. The first one is that taking the expectation over all possible assignments $\mathbf{Y}$ in Equation 10 is computationally expensive: The assignments will be to all subsequences of atoms in the training data. The second, more serious problem is that there is no closed form solution to the maximization step, because the value of the cost function changes discontinuously with the number of constructions that are stored in the lexicon. Testing all possible lexicons—that is, all subsets of all subsequences of atoms—is clearly infeasible. Another type of a prior distribution might enable forward-backward training, but the current implementation supports only the MDL-style prior described in Section 2.1.

Global Viterbi Algorithm

A simple approximation to the forward-backward algorithm is to first take the most probable analysis $\phi_{\text{best}}(\boldsymbol{w}; \boldsymbol{\theta}^{(t-1)})$ for each compound and then update the parameters to minimize the cost function:

$$\boldsymbol{\theta}^{(t)} = \arg\min_{\boldsymbol{\theta}} \Big\{ -\log p(\boldsymbol{\theta}) - \log \prod_{j=1}^{|\boldsymbol{D}_W|} p\big(\phi_{\text{best}}(\boldsymbol{w}_j; \boldsymbol{\theta}^{(t-1)}) \mid \boldsymbol{\theta}\big) \Big\}, \tag{12}$$

where

$$\phi_{\text{best}}(\boldsymbol{w}; \boldsymbol{\theta}) = \arg\max_{\boldsymbol{a}} p(\boldsymbol{a} \mid \boldsymbol{w}, \boldsymbol{\theta}) = \arg\max_{\substack{m_1,\ldots,m_n: \\ \boldsymbol{w}=m_1\ldots m_n}} p(m_1, \ldots, m_n, \#_w \mid \boldsymbol{\theta}). \tag{13}$$

The best segmentation can be solved by a generalization of the Viterbi algorithm for hidden Markov models (Viterbi, 1967; Forney, 1973). Here, the observation is the sequence of $|\boldsymbol{w}|$ atoms that form the compound $\boldsymbol{w}$, and the hidden states are the constructions of the compound. In contrast to the

standard Viterbi, one state (construction) can overlap several observations (atoms). When selecting the best path to the ith observation, it is possible to come from any observation between one and $i-1$. This increases the time complexity of the algorithm by a factor of $|\boldsymbol{w}|$, thus giving complexity of $O(|\boldsymbol{w}|^2)$.

The main problem of the Viterbi algorithm is that it cannot assign a non-zero probability to any construction that was not stored in the previous lexicon. As the construction lexicon can only be reduced, the initialization has a huge impact on the results.

The Viterbi search for Equation 13 is used also as a tokenization algorithm. That is, it finds the most likely analyses for new compounds after the model parameters have been set in the actual training phase.

Viterbi tokenization is supported both in Morfessor 1.0 and 2.0. Given a reasonably good initial segmentation, it can also be used to optimize the parameters: The parameters are first estimated from the known segmentation of the training data. Then the training data is retokenized with the Viterbi algorithm. These two steps can be repeated until there is no change in the segmentation result.

In addition to the generalized Viterbi algorithm described above, Morfessor 2.0 includes the equivalent Forward algorithm, which calculates the probability of a compound given the model, and n-best Viterbi decoding, which provides the n most likely tokenizations for a compound.

Local Viterbi Algorithm

Alternatively to updating the parameters globally in one step, the Viterbi algorithm can also be applied locally to one compound at a time: First, the optimal segmentation $\phi_{\text{best}}(\boldsymbol{w};\boldsymbol{\theta})$ is searched and then the parameters are updated according to the new segmentation.

The Morfessor 2.0 implementation includes local Viterbi training. To alleviate the problems non-zero probabilities, additive smoothing may be applied to the probabilities used in Viterbi search. Given the smoothing constant $\lambda > 0$, the probability of a construction m_i that is already in the lexicon is estimated by

$$p_{\text{old}}(m_i \mid \boldsymbol{\theta}) = \frac{\tau_i + \lambda}{\nu + \lambda\mu}. \tag{14}$$

Moreover, for the probability of a construction m that is not in the lexicon is set to

$$p_{\text{new}}(m \mid \boldsymbol{\theta}) \approx \frac{\lambda}{\nu + \lambda\mu} \times \frac{p(\tilde{\boldsymbol{\theta}})p(\boldsymbol{D}_W \mid \tilde{\boldsymbol{\theta}})}{p(\boldsymbol{\theta})p(\boldsymbol{D}_W \mid \boldsymbol{\theta})}. \tag{15}$$

Here $p(\tilde{\boldsymbol{\theta}})$ and $p(\boldsymbol{D}_W \mid \tilde{\boldsymbol{\theta}})$ are approximated prior and likelihood probabilities in the case that m is added to the lexicon (Virpioja et al., 2010; Kohonen et al., 2010a; Virpioja et al., 2011a). For example, if the proper noun **matthew** was never observed in the training data, it would likely to be oversegmented by the standard Viterbi (e.g. **m+at+the+w**). If $p_{\text{new}}(\textbf{matthew} \mid \boldsymbol{\theta})$ was higher than the likelihood of the segmentation, the augmented Viterbi would leave the compound intact.

While the smoothed Viterbi training is, in principle, able to introduce new constructions, it is still a very conservative algorithm: It does not take into account that adding a new construction to the lexicon is likely to increase likelihoods of many compound forms, not just the current compound.

Recursive Baseline Algorithm

The standard training algorithm for Morfessor Baseline—described in detail also by Creutz and Lagus (2005b)—applies a recursive, greedy search. At each step, changes that modify only a small part of the parameters are considered, and the change that returns the minimal cost is selected. Similar to the Viterbi approach, only one potential analysis $\boldsymbol{y}_j \in \Phi(\boldsymbol{w}_j)$ is set to be active at a time. In consequence, a zero probability will be assigned to a large part of the potential constructions. As they do not have to be stored in the lexicon, this type of an algorithm is very memory-efficient.

In the simplest case, the local optimization algorithm considers one compound $\boldsymbol{w}_j$ at a time. First, the analysis that minimizes the cost function with the optimal model parameters is selected:

$$\boldsymbol{y}_j^{(t)} = \arg\min_{\boldsymbol{y}_j \in \mathcal{Y}_j} \left\{ \min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \mathbf{Y}^{(t-1)}, \boldsymbol{D}_W) \right\}. \tag{16}$$

Then the parameters are updated:

$$\boldsymbol{\theta}^{(t)} = \arg\min_{\boldsymbol{\theta}} \left\{ L(\boldsymbol{\theta}, \mathbf{Y}^{(t)}, \boldsymbol{D}_W) \right\}. \tag{17}$$

As neither of the two steps can increase the cost function, the algorithm will converge to a local optimum.

The training algorithm of Morfessor Baseline exploits the assumption that the constructions occur independently. As the optimal analysis of a segment is context-independent, one optimization step can modify segments shared by multiple compounds and not only one as in Equations 16–17.

In order to efficiently exploit the context-independence, the analyses $\mathbf{Y}$ are stored in a binary directed acyclic graph. The leaf nodes of the graph are constructions. All other nodes can be considered as "virtual constructions"—they are not stored by the model parameters, but help keeping track of the

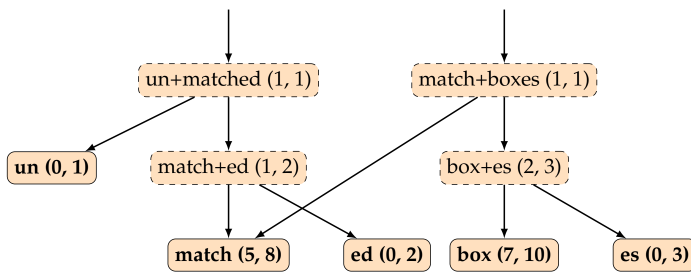


Figure 2. Example of an analysis graph in the recursive training algorithm for words **unmatched**, **matchboxes**, **matched**, **boxes**, **match**, and **box**.

analyses and counts. The top nodes are always compounds, but also any other node can be a compound.

Apart from its possible children, each node stores two counts: *root count* and *total count*. The former gives how many times the node occurs as a compound, and the latter gives the total count of references to the node. For any node, the total count is the sum of its root count and the total counts of its immediate parents, if any. Thus, for top nodes, the total count equals the root count. The total count of a leaf node gives how many times the respective construction in applied in the analyses of the training data (τ_i). For example, in the analysis graph shown in Figure 2, **match** is applied eight times: once in **matchboxes**, once in **unmatched**, once in **matched**, and five times in **match**.

In the recursive baseline search, the local optimization step modifies the nodes of the graph: for the current node, it considers every possible split into two constructions, as well as no split. If the node is split, the search is applied recursively to its child nodes. Outline for the RECURSIVESEARCH procedure is shown in Algorithm 4.

Algorithm 4 Recursive baseline algorithm for local search. $\boldsymbol{w}[i \dots j]$ denotes the atoms of $\boldsymbol{w}$ from the ith to the jth atom. $\textsc{Node}(a, b)$ creates a new node with children a and b and $\textsc{LeafNode}()$ creates a new leaf node.

```
function RecursiveSearch(w, D, θ, Y)
    Y[w] ← LeafNode()
    θ ← arg min_θ L(D_W, θ, Y)
    l_min ← L(D, θ, Y)                          ▷ Best cost so far
    i_min ← 0                ▷ Best split point so far (0 = no split)
    n ← |w|
    for i ∈ {1, 2, ..., n} do                  ▷ Test possible split points
        Y[w] ← Node(w[1...i], w[(i+1)...n])
        θ ← arg min_θ L(D_W, θ, Y)
        if L(D, θ, Y) < l_min then
            l_min ← L(D, θ, Y)
            i_min ← i
        end if
    end for
    if i_min = 0 then                         ▷ No split: add as a leaf node
        Y[w] ← LeafNode()
        θ ← arg min_θ L(D_W, θ, Y)
    else                      ▷ Split at i_min, add the node and recurse
        Y[w] ← Node(w[1...i_min], w[(i_min+1)...n])
        θ ← arg min_θ L(D_W, θ, Y)
        θ, Y ← RecursiveSearch(w[1...i_min], D_W, θ, Y)
        if w[1...i_min] ≠ w[(i_min+1)...n] then
            θ, Y ← RecursiveSearch(w[(i_min+1)...n], D_W, θ, Y)
        end if
    end if
    return θ, Y
end function
```

In practice, the graph ($\mathbf{Y}$) and the parameters ($\boldsymbol{\theta}$) are updated simultaneously by using the construction counts in the analysis graph. Constructions are then added to or removed from the lexicon when their counts increase above zero or decrease to zero, respectively. Moreover, as direct calculation of the data likelihood is too slow to be done many times during each call of RecursiveSearch, the essential part of the log-likelihood value is kept up-to-date when modifying the graph. Given the counts of the constructions τ_i,

we can write the log-likelihood as a sum over the construction types:

$$\begin{aligned}
\log p(\boldsymbol{D}_W \mid \boldsymbol{\theta}, \mathbf{Y}) &= \sum_{i=1}^{\mu} \tau_i \log \frac{\tau_i}{N+\nu} + N \log \frac{N}{N+\nu} \\
&= \sum_{i=1}^{\mu} \tau_i (\log \tau_i - \log(N+\nu)) + N(\log N - \log(N+\nu)) \\
&= \sum_{i=1}^{\mu} \tau_i \log \tau_i - \sum_{i=1}^{\mu} \tau_i \log(N+\nu) + N \log N - N \log(N+\nu) \\
&= \sum_{i=1}^{\mu} \tau_i \log \tau_i + N \log N - (N+\nu) \log(N+\nu) \qquad (18)
\end{aligned}$$

Thus, if we keep track of the first term ("log-token sum") and ν, and update them whenever the construction counts are modified, calculation of the log-likelihood is very efficient.

Training Speed-Up with Skips

Because of its recursive nature, the standard training algorithm tests common constructions—such as affix morphemes in morphological segmentation—very frequently. Moreover, in on-line training scheme, the most frequent compounds are observed repeatedly. Because the analysis of frequently seen compounds and constructions is unlikely to change very often, a useful trick is to collect statistics on how often each virtual construction is tested, and skip the recursive search with a probability that decreases as a function of the number of tests.

Let $s(\boldsymbol{w})$ be the number of times $\boldsymbol{w}$ has been tested by RECURSIVESEARCH. The implemented speed optimization skips $\boldsymbol{w}$ with the probability

$$p(\text{skip} \mid \boldsymbol{w}) = 1 - \frac{1}{\max(1, s(\boldsymbol{w}))}. \qquad (19)$$

If $\boldsymbol{w}$ is skipped, the counter $s(\boldsymbol{w})$ is not increased. Thus, if $\boldsymbol{w}$ has so far been tested 100 times, it is likely to be tested at least once during the next 100 tries.

To ensure that the skips do not hinder the optimization task by making it hard to change the analysis of frequent constructions, the counters are reset periodically. In batch training, it is done after every training epoch. In on-line training, it is done after processing a fixed amount of compounds ("epoch interval"; the default value is set to 10 000).

Likelihood Weights and Semi-Supervised Training

Morfessor Baseline tends to undersegment when the model is trained for morphological segmentation using a large corpus (Creutz and Lagus, 2005b, 2007). Oversegmentation or undersegmentation of the method are easy to control heuristically by including a weight parameter $\alpha > 0$ for the likeli-

hood in the cost function (Kohonen et al., 2010b; Virpioja et al., 2011a):

$$L(\boldsymbol{\theta}, \boldsymbol{D}_W) = -\log p(\boldsymbol{\theta}) - \alpha \log p(\boldsymbol{D}_W \mid \boldsymbol{\theta}). \tag{20}$$

A low α means that most of the cost comes from the prior, and thus small construction lexicons are favored. A high α means that the cost is dominated by the likelihood, and thus long constructions are favored.

In semi-supervised Morfessor (Kohonen et al., 2010b), the likelihood of an annotated data set $\boldsymbol{D}_{W\to A}$ is added to the cost function. As the amount of annotated data is typically much lower than the amount of unannotated data, its effect on the cost function may be very small compared to the likelihood of the unannotated data. To control the effect of the annotations, a separate weight parameter $\beta > 0$ can be included for the annotated data likelihood:

$$L(\boldsymbol{\theta}, \boldsymbol{D}_W) = -\log p(\boldsymbol{\theta}) - \alpha \log p(\boldsymbol{D}_W \mid \boldsymbol{\theta}) - \beta \log p(\boldsymbol{D}_{W\to A} \mid \boldsymbol{\theta}). \tag{21}$$

If separate development data set is available for automatic evaluation of the model, the likelihoods weights can be optimized to give the best output. This can be done by brute force using a grid search. However, Morfessor 2.0 implementation includes a simple method for automatically tuning the value of α during the training.

Let us assume that our evaluation metric provides precision P and recall R such that $P > R$ indicates undersegmentation and $R > P$ indicates over-segmentation.[2] Between each training epoch, we calculate P and R. Let $d = \mathrm{sign}(R - P)$ if the difference between P and R is larger than a given threshold or $d = 0$ otherwise. We set

$$\alpha \to \alpha \times (1 + 2/e)^d, \tag{22}$$

where e is the number of epochs passed so far. Thus, α can either be multiplied or divided by 3 after the first epoch, by 2 after the second epoch, and so forth.

As updating the value of α changes the value of the cost function, convergence testing is skipped until two training epochs have passed without changes to α.

Tuning both α and β between the training epochs is difficult, as they may interact. For β, we use by default a simple heuristic and set $\beta = \alpha \frac{|\boldsymbol{D}_W|}{|\boldsymbol{D}_{W\to A}|}$, so that the both data sets will have a similar contribution to the cost function regardless of their size.

[2] Apart from simply calculating the precision and recall of the boundary positions, there are also more indirect evaluation measures that have this property (Virpioja et al., 2011b).

Forced Splitting and Joining

Finally, Morfessor 2.0 includes two options for manually controlling the segmentation behavior. First, the user can list a set of atoms that will always be split to constructions of their own. In the default setting, intended for morphological segmentation, this list includes the hyphen. Other useful characters may be apostrophes and colons, depending on the target language. In chunking, it may be useful to always split at punctuation characters such as comma and semicolon.

Second, the user can provide a regular expression that prevents segmentation for any position for which the regular expression matches the two surrounding characters. This option is valid only if the atoms are characters. It can be used, for example, to force that all non-alphabetic characters are joined together.

3 Evaluation

A new feature of Morfessor 2.0 is the integration of the standard evaluation and model comparison methods. In particular, we have implemented micro-average segmentation boundary precision/recall evaluation on a gold-standard data set for evaluation and the Wilcoxon signed-rank test for statistical significance testing.

3.1 Boundary Precision and Recall Evaluation

For evaluating a segmentation task, we have implemented an algorithm for calculating the micro-average segmentation boundary precision, recall, and F-score measures.

The precision and recall metrics are applied to the boundary predictions, with a gold-standard file for reference. This makes the definitions:

$$\text{precision} = \frac{\text{number of correct boundaries found}}{\text{total number of boundaries found}} \tag{23}$$

$$\text{recall} = \frac{\text{number of correct boundaries found}}{\text{total number of correct boundaries}} \tag{24}$$

The F-score is the harmonic mean of the precision and the recall:

$$\text{F-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{25}$$

The algorithm is equivalent to the "BPR" evaluation described in Virpioja et al. (2011b) except for how the alternative tokenizations for the same compound are handled. We expect only one tokenization per compound from

the evaluated model. In the case of alternative tokenizations for the same compound in the gold standard, we select the one that provides the highest precision or recall. An earlier BPR implementation that includes more evaluation options is available from http://research.ics.aalto.fi/events/morphochallenge/.

3.2 Statistical Significance Testing

To test whether the performance of two models differ significantly the Wilcoxon signed-rank test (Wilcoxon, 1945) has been implemented. This method is a non-parametric, paired difference test. In order for it to be in line with modern standards, the ranking method has been updated with the treatment from Pratt (1959) and a correction of 0.5 towards the mean value is applied. The p-value is reported for interpretation by the user.

4 Experiments

We present a set of experimental results for Morfessor 2.0 in English and Finnish morphological segmentation tasks. First, we show how the new implementation compares to the Morfessor Baseline 1.0 implementation. Then we show how the additional features of the Morfessor 2.0 implementation change the results compared to the baseline results.

Data Sets

As the data, we use the English and Finnish data sets from Morpho Challenge 2010 (Kurimo et al., 2010). Table 3 shows the number of samples in the data sets. Except for the test sets, the data sets are available from http://research.ics.aalto.fi/events/morphochallenge2010/datasets.shtml.

Table 3. The numbers of word types in the English and Finnish Morpho Challenge 2010 data sets Kurimo et al. (2010).

	English	Finnish
Unannoted training set	878 036	2 928 030
Annotated training set	1 000	1 000
Annotated development set	694	835
Test sets	10×1 000	10×1 000

For the experiments with on-line training, we use the unannotated training corpora from Challenge 2007, available from http://research.ics.aalto.fi/events/morphochallenge2007/datasets.shtml. We filter out all tokens

of the corpus that are not included in the corresponding word list. The cleaned corpora contain in 62 385 521 tokens for English and 36 440 171 tokens for Finnish. These smaller data sets are also used in the semi-supervised training experiments.

Evaluation

As the evaluation metric, we use the micro-average segmentation boundary F-score as described in Section 3.1. The scores are calculated over the word types in the evaluation sets.

We report development set results for all the experiments and the test set results only for those cases where the development set is used for hyperparameter optimization. The words in the evaluation sets are tokenized with the Viterbi algorithm after training the model.

4.1 Comparison to Morfessor 1.0

We compare Morfessor 2.0 to Morfessor 1.0 on a number of different aspects. Besides formal final results in segmentation quality we also evaluate runtime and convergence speed.

Runtime comparison between Morfessor 1.0 and Morfessor 2.0

On a modern 64-bit Linux system we have tested the runtime speed for a full unsupervised, type based, training of the Finnish and English datasets. As there are three compatible Python interpreters available for this platform we evaluated Morfessor 2.0 with all three.

In Table 4 the runtime in seconds is reported for all configurations. For the English training with the standard Python interpreters for Morfessor 2.0 the training is 2–8% slower than the Perl 1.0 version. Analysis show that this small degradation is caused by the full unicode implementation, which causes a double in memory usage and slower memory access times.[3] Looking to the same results for Finnish confirms this theory, as the bigger Finnish dataset causes a 104% slowdown.

However, there are also alternative Python interpreters available. Morfessor 2.0 is compatible with the PyPy interpreter[4], which includes an integrated just-in-time compiler that optimizes the code runtime. With PyPy, training is

[3] Python version 3.3 includes a feature that reduces the memory usage for unicode strings if only a specific subset is used, e.g. ansi or latin-1 (`http://www.python.org/dev/peps/pep-0393/`). This version will most likely speed up the training, but was not yet available on our platform for testing.

[4] `http://pypy.org`

Table 4. Training speed comparison Morfessor 1.0 and Morfessor 2.0 with default settings.

Version	Interpreter	Eng/Time (s)	Fin/Time (s)
1.0	Perl	4339	18720
2.0	Python 2.7	4410	35160
2.0	Python 3.2	4680	38160
2.0	Pypy 1.9	**810**	**5700**

Table 5. Difference in result between Morfessor 1.0 and Morfessor 2.0 (fs = forced splitting of hyphens, nfs = no forced splits).

Run	Epochs	Time (s)	Cost (nats)	Pre. (%)	Rec. (%)	F-s. (%)
English						
1.0	6	4339	20861267	0.89	0.67	0.76
2.0, fs	5	810	21281230	0.89	0.67	0.76
2.0, nfs	6	1385	20861747	0.90	0.64	0.74
Finnish						
1.0	5	18730	73964660	0.86	0.43	0.57
2.0, fs	5	5700	74360279	0.87	0.42	0.57
2.0, nofs	5	6300	73951063	0.87	0.41	0.56

much faster—81% for English and 70% for Finnish—than with the Morfessor 1.0 implementation.

Comparing the final results of Morfessor 1.0 and Morfessor 2.0

As some details of the algorithm and default settings have changed between Morfessor 1.0 and Morfessor 2.0, we first compare their results in unsupervised, type-based training.

The results for English and Finnish are shown in Table 5. Morfessor 2.0 was tested on two different configurations, with and without the forced split option. The default configuration is to use forced splitting for the hyphen character. Morfessor 1.0 does not support forced splitting.

There is no significant difference in results between Morfessor 1.0 and 2.0. The slightly lower recall of Morfessor 2.0 without forced splits is an effect of the changes in handling the word boundaries (see Section 2.1). The costs of the models are almost equal. With forced splits, the F-scores of Morfessor 2.0 increase to the same level as Morfessor 1.0, but the costs are somewhat higher.

Convergence

The results in Table 5 show a difference in the number of training epochs for English and the final cost between Morfessor 2.0 with forced splits and Morfessor 1.0 and 2.0 without forced splits. By looking at the convergence

we show that the general behavior is still the same and the stopping criterion reasonable.

Figure 3 shows the convergence results for the model cost. For both English and Finnish, the cost of the models start to stabilize after the third iteration. However, they do not fully converge even after 10 iterations. The forced splitting prevents obtaining as low cost as without it. Without forced splitting, Morfessor 2.0 reaches a similar level of model cost as Morfessor 1.0.

The convergence of the F-score is shown in Figure 4. For English, the F-scores converge after 4–5 epochs for all runs. The stopping criterion interrupts the algorithm after 5 epochs and thus its default threshold parameter is, in this case, optimal. For Finnish, the F-scores converge after 3–4 epochs. The final number of epochs of 5 is more than necessary, so the convergence criterion could even be slightly relaxed.

Figure 3. Convergence of model cost for Morfessor 1.0 (m1), Morfessor 2.0 (m1.fs) and Morfessor 2.0 without force-splitting the hyphen character (m1.nfs).

(a) English **(b)** Finnish

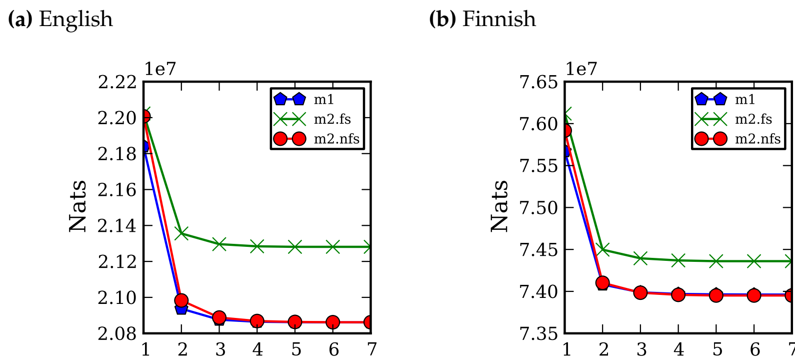
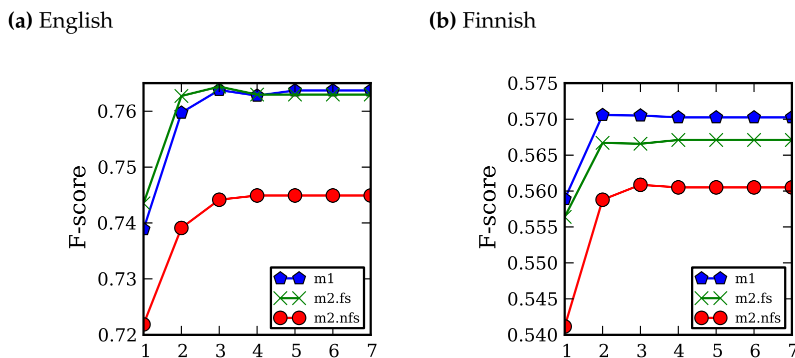


Figure 4. Convergence of F-scores for Morfessor 1.0 (m1), Morfessor 2.0 (m1.fs) and Morfessor 2.0 without force-splitting the hyphen character (m1.nfs).

(a) English **(b)** Finnish



4.2 Random Variation in Model Training

In order to observe the effect of random variation in the standard recursive training algorithm, we ran the training with ten different random seeds for both English and Finnish. The mean and standard deviation of the training times, cost function values, and boundary evaluation scores are shown in Table 6. The absolute standard deviation of F-score was similar, 0.34%, for both languages. The deviation of the cost function was about 0.3% relative for both languages.

Table 6. Random variation in model training. (PyPy 1.8 interpreter, Morpho Challenge 2010 training data, development set scores.)

Run	Epochs	Time (s)	Cost (nats)	Pre. (%)	Rec. (%)	F-s. (%)
English						
mean	5	558	21281456	89.95	66.38	76.39
st. dev	0	28	651	0.45	0.29	0.33
Finnish						
mean	5	3613	74357060	86.25	41.97	56.46
st. dev	0	91	2307	0.46	0.27	0.33

4.3 Random Split Initialization

We tested the random split initialization with a few values between 0.1 and 1.0. It had only minor effect to the segmentation results (Table 7). A low, non-zero probability of splitting works actually worse than no splitting: training time is increased, but there is no gain in cost function or scores. In contrast, for $p \geq 0.5$, the cost function had always lower value than for $p = 0$, which indicates that random initialization helps finding better local optimums. The downside is about 20% increased training time.

4.4 Training Speed-up with Skips

Table 8 compares the batch training results with random skipping of frequent constructions. Skipping provided around 30% decrease in training times for both English and Finnish task. Difference to the original F-score is smaller than the standard deviation due to the random initialization of the training algorithm, so the speed-up comes with no practical cost.

In on-line training, the random skipping is even more useful, because frequent words are encountered all over again. The speed-up is 34% for English and 29% for Finnish and the evaluation scores only increase, when the skip

Table 7. The effect of random split initialization. (PyPy 1.8 interpreter, Morpho Challenge 2010 training data, development set scores.)

Run	Epochs	Time (s)	Cost (nats)	Pre. (%)	Rec. (%)	F-s. (%)
English						
$p = 0.0$	5	536	21281287	**89.55**	66.47	76.30
$p = 0.1$	5	541	21295993	86.79	66.01	74.99
$p = 0.2$	5	592	21283454	86.02	67.10	75.39
$p = 0.5$	5	653	**21267223**	87.15	67.08	75.81
$p = 0.8$	5	619	21272590	88.14	68.24	76.93
$p = 0.9$	5	591	21273485	89.12	**68.53**	**77.48**
$p = 1.0$	5	651	21276437	88.45	67.10	76.31
Finnish						
$p = 0.0$	5	3810	74351550	**86.47**	41.79	56.34
$p = 0.1$	6	4616	74591670	81.46	40.37	53.99
$p = 0.2$	6	6743	74447555	82.48	41.18	54.93
$p = 0.5$	5	4075	74322825	85.29	**42.09**	**56.37**
$p = 0.8$	5	4248	**74281444**	84.43	41.55	55.69
$p = 0.9$	5	4211	74289057	84.49	41.65	55.80
$p = 1.0$	5	4513	74286697	84.85	41.78	55.99

Table 8. The effect of random skipping in batch training. (PyPy 1.8 interpreter, Morpho Challenge 2010 training data, development set scores.)

Run	Epochs	Time (s)	Cost (nats)	Pre. (%)	Rec. (%)	F-s. (%)
English						
no skips	5	536	21281287	89.55	66.47	76.30
skips	5	358	21282049	88.88	65.91	75.69
Finnish						
no skips	5	3810	74351550	86.47	41.79	56.34
skips	5	2674	74362175	86.23	41.75	56.26

counters were reset after each 10 000 words (Table 8). Larger values of the interval parameter increase the speed further, as reported later in Section 4.6.

4.5 Viterbi Training

Due to the conservative nature of local Viterbi training, it is not very useful as such. Initializing the model lexicon with words results in high precision but low recall (Table 10). With random split initialization, more balanced precision and recall can be obtained, but the scores are much worse than those obtained from the recursive baseline algorithm. Increasing the additive smoothing constant λ (Equation 14) had only negligible effect to the results.

When applied after the recursive training algorithm, the Viterbi algorithm

Table 9. The effect of random skipping in on-line training with the default epoch interval parameter (10 000). (PyPy 1.8 interpreter, Morpho Challenge 2007 training data, development set scores.)

Run	Epochs	Time (s)	Cost (nats)	Pre. (%)	Rec. (%)	F-s. (%)
English						
no skips	$\frac{N}{10^4}$	3472	9063019	78.65	73.41	75.94
skips	$\frac{N}{10^4}$	2279	9063313	79.43	74.52	76.90
Finnish						
no skips	$\frac{N}{10^4}$	4871	55982728	81.46	43.63	56.83
skips	$\frac{N}{10^4}$	3477	55985301	82.30	44.10	57.42

was able to decrease the cost function slightly (0.1%). This does not show in the boundary scores—probably because the test set is in any case segmented with the Viterbi algorithm.

Training times for the local Viterbi are less than half of the times for the recursive algorithm.

Table 10. Results of local Viterbi training. (PyPy 1.8 interpreter, Morpho Challenge 2010 training data, development set scores.)

Run	Epochs	Time (s)	Cost (nats)	Pre. (%)	Rec. (%)	F-s. (%)
English						
recursive	5	536	21281287	89.55	66.47	76.30
rec.+Viterbi	5+2	619	21256231	89.55	66.47	76.30
Viterbi	2	76	25978025	99.57	23.49	38.01
Viterbi, $p = 0.1$	4	171	25683288	76.20	42.56	54.62
Viterbi, $p = 0.2$	4	188	25498304	52.12	47.59	49.75
Viterbi, $p = 0.5$	3	136	26067407	22.34	60.91	32.69
Finnish						
recursive	5	3810	74351550	86.47	41.79	56.34
rec.+Viterbi	5+2	4263	74273417	86.47	41.79	56.34
Viterbi	2	356	130269229	99.96	5.03	9.58
Viterbi, $p = 0.1$	6	1437	103684416	64.09	23.64	34.54
Viterbi, $p = 0.2$	6	1471	94803114	39.03	24.78	30.31
Viterbi, $p = 0.5$	5	3861	95284200	26.94	39.67	32.09

4.6 On-line Training

Next, we compare on-line training, batch training, and their combination, in which batch training is applied after the on-line training has processed the whole training corpus. We use random skipping in the training regardless of the training scheme.

The results are shown in Table 11. In on-line training, the training time is

affected by the number of tokens, not types. Accordingly, on-line training is relatively quicker on the Finnish data set, which has a lower token-to-type ratio than the English data set. With large enough epoch interval, the on-line training with skips is in fact quicker than the batch training.

On-line training alone does not provide as low a model cost as batch training, but their combination always gives lower cost than using only batch training. However, the lower cost does not lead to higher F-scores. For English, recall scores are higher but precision scores lower than with batch training. For Finnish, even the recall scores do not clearly improve.

Table 11. The results of on-line training. Random skipping was used for all runs. (PyPy 1.8 interpreter, Morpho Challenge 2007 training data, development set scores.)

Type	Epochs	Time (s)	Cost (nats)	Pre. (%)	Rec. (%)	F-s. (%)
English						
batch	- / 6	217	9026845	**82.46**	71.44	76.55
online	$\frac{N}{10^4}$ / -	2279	9063313	79.43	74.52	**76.90**
online+batch	$\frac{N}{10^4}$ / 3	2334	9018151	79.43	74.34	76.80
online	$\frac{N}{10^5}$ / -	1219	9062868	78.75	73.50	76.03
online+batch	$\frac{N}{10^5}$ / 3	1408	9018035	78.67	73.65	76.08
online	$\frac{N}{10^6}$ / -	670	9063684	78.53	73.60	75.99
online+batch	$\frac{N}{10^6}$ / 3	823	9017490	78.73	73.34	75.94
online	$\frac{N}{10^7}$ / -	452	9069958	78.54	74.77	76.61
online+batch	$\frac{N}{10^7}$ / 3	546	**9017458**	78.64	74.03	76.27
online	$\frac{N}{10^8}$ / -	413	9201149	75.72	**76.13**	75.92
online+batch	$\frac{N}{10^8}$ / 4	538	9017652	78.87	74.16	76.44
Finnish						
batch	- / 5	1870	55591891	**85.11**	44.44	**58.39**
online	$\frac{N}{10^4}$ / -	3477	55985301	82.30	44.10	57.42
online+batch	$\frac{N}{10^4}$ / 3	4463	55557045	82.59	43.67	57.13
online	$\frac{N}{10^5}$ / -	2583	55987962	82.42	43.93	57.31
online+batch	$\frac{N}{10^5}$ / 3	3716	55552566	82.71	43.70	57.18
online	$\frac{N}{10^6}$ / -	1673	55995437	82.71	44.21	57.62
online+batch	$\frac{N}{10^6}$ / 3	2728	55551577	83.03	43.83	57.37
online	$\frac{N}{10^7}$ / -	1039	56064939	81.76	43.58	56.86
online+batch	$\frac{N}{10^7}$ / 4	2544	55548172	82.38	43.09	56.58
online	$\frac{N}{10^8}$ / -	789	56688964	78.88	**44.48**	56.88
online+batch	$\frac{N}{10^8}$ / 4	2208	**55542310**	82.02	43.03	56.44

4.7 Frequency Dampening

The frequency dampening effects (cf. Virpioja et al., 2011a) are shown in Table 12. If the word frequencies affect the likelihood, the recall decreases

and the precision increases, as common strings are left unsegmented. When trained with word tokens, the number of epochs needed for convergence is about 50% less than that are needed with dampened frequencies.

Table 12. The effect of frequency dampening. (PyPy 1.8 interpreter, Morpho Challenge 2010 training data, development set scores.)

Data	Epochs	Time (s)	Cost (nats)	Pre. (%)	Rec. (%)	F-s. (%)
English						
types	5	536	21281287	89.55	66.47	76.30
logarithmic	5	527	41599429	97.84	50.55	66.66
tokens	2	221	2025408238	97.19	39.23	55.90
Finnish						
types	5	3810	74351550	86.47	41.79	56.34
logarithmic	5	3505	122245184	90.72	36.87	52.43
tokens	3	2157	800405681	93.41	29.69	45.06

4.8 Weight Optimization

Changing of the likelihood weight makes it possible to optimize the balance between precision and recall. Typically the precision is higher and recall is lower the larger the unannotated training data is, so the weighting can compensate for training corpora that do not happen to be of optimal size for the desired target. The weighting is even more useful if the word frequencies are not dampened Virpioja et al. (2011a).

Table 13 shows the result for optimizing the weight automatically for the annotated development data set as described in Section 2.2. We have trained the models with word types, first from the full Morpho Challenge 2010 data sets and then from subsets of 100 000 sentences. For the full English data set, the default weight (1.0) is already near the optimum, and the automatic balancing actually degrades the test set F-score. For the smaller data set, optimization increases F-score 2.6% absolute. For Finnish, 4.2% absolute increase is obtained for the full data set and 1.6% for the smaller data set.

4.9 Semi-supervised Training

Semi-supervised training for Morfessor (Kohonen et al., 2010a,b) has various options considering how the likelihood weights are selected. The results are shown in Table 14. Without weighting, the effect of the small annotated data set is very small. However, already the heuristic value for β improves the situation remarkably, providing over 10% increase in F-score for English and

Table 13. The effect of weight optimization. (PyPy 1.8 interpreter, Morpho Challenge 2010 training data, development set and test set scores.)

Run	Epochs	Time (s)	α	Pre. (%)	Rec. (%)	F-s. (%)
English, full data (878 000 word types)						
unsupervised	5	590	1.000	89.67/**87.82**	66.05/66.64	76.07/**75.77**
optimized	7	942	0.298	73.97/70.63	75.59/**75.59**	74.77/73.02
English, subset (73 000 word types)						
unsupervised	7	62	1.000	70.50/65.80	79.33/**77.97**	74.65/71.37
optimized	11	97	1.331	75.68/**71.40**	76.79/76.75	76.23/**73.98**
Finnish, full data (2 928 000 word types)						
unsupervised	5	4071	1.000	86.37/**83.42**	42.51/41.17	56.97/55.12
optimized	14	13968	0.013	62.60/58.13	64.96/**60.51**	63.76/**59.29**
Finnish, subset (243 000 word types)						
unsupervised	6	343	1.000	74.23/**70.46**	52.71/48.96	61.64/57.77
optimized	8	510	0.158	62.30/58.88	63.34/**59.80**	62.81/**59.33**

24% for Finnish. On-line optimization of α provides some further improvement for Finnish but not for English. The final rows in Table 14 show the optimal values of α and β found using a manual grid search. The results show that there is room for improvements for the automatic selection of the weights. Especially for Finnish, the heuristic value of β is much smaller than the optimal value. Thus testing a few values manually is recommended.

Table 14. Semi-supervised training. (PyPy 1.8 interpreter, Morpho Challenge 2010 training data, development set and test set scores.)

Run	Epochs	Time (s)	α	β	Pre. (%)	Rec. (%)	F-s. (%)
English							
no supervision	5	590	1.000	1	89.67/87.82	66.05/66.64	76.07/75.77
no weighting	5	636	1.000	1	89.89/87.98	66.48/66.76	76.43/75.90
heuristic β	5	710	1.000	878	89.00/86.97	81.88/80.98	85.29/83.86
opt. α, heur. β	5	739	0.667	585	83.32/81.51	83.66/83.64	83.49/82.56
grid search	4	592	1.000	3000	85.91/83.37	85.50/85.10	85.71/84.22
Finnish							
no supervision	5	4071	1.000	1	86.37/83.42	42.51/41.17	56.97/55.12
no weighting	5	4107	1.000	1	86.67/83.56	42.66/41.27	57.17/55.25
heuristic β	2	2195	1.000	2928	86.64/84.00	59.20/57.60	70.34/68.33
opt. α, heur. β	10	11415	0.022	65	71.76/67.42	74.05/71.33	72.89/69.32
grid search	2	2490	0.200	15000	78.68/76.06	78.42/77.03	78.55/76.53

5 Conclusions

In this report, we have described a number of corrections, improvements and extensions for the Morfessor Baseline method and verified their effects on English and Finnish morphological segmentation tasks. All new features are available in our Python implementation, Morfessor 2.0. We have aimed for a software package that is as simple to use as the original Perl implementation of Morfessor Baseline, but more robust, efficient and easier to extend further. We hope that it serves both those who want to integrate Morfessor to their natural language processing applications and those who are interested in developing the methodology further.

Morfessor 2.0 software package is available under a permissive FreeBSD-style license at http://www.cis.hut.fi/projects/morpho/ or from GitHub repository at https://github.com/aalto-speech/morfessor.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007–2013) under grant agreement n°287678 and the Academy of Finland under the Finnish Centre of Excellence Program 2012–2017 (grant n°251170) and the LASTU Programme (grants n°256887 and 259934). The experiments were performed using computer resources within the Aalto University School of Science "Science-IT" project. We are grateful to Oskar Kohonen for his help with the experiments and the initial Python implementations of Morfessor. We also thank Dr. Mathias Creutz, Dr. Krista Lagus, Matti Varjokallio, and Teemu Ruokolainen for their comments and many valuable discussions.

Bibliography

Arisoy, E., Can, D., Parlak, S., Sak, H., and Saraclar, M. (2009). Turkish broadcast news transcription and retrieval. *Audio, Speech, and Language Processing, IEEE Transactions on*, 17(5):874–883.

Baum, L. E. (1972). An inequality and an associated maximization technique in statistical estimation of probabilistic functions of a Markov process. *Inequalities*, 3(1):1–8.

Clifton, A. and Sarkar, A. (2011). Combining morpheme-based machine translation with post-processing morpheme prediction. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 32–42, Portland, Oregon, USA. Association for Computational Linguistics.

Creutz, M., Hirsimäki, T., Kurimo, M., Puurula, A., Pylkkönen, J., Siivola, V., Varjokallio, M., Arisoy, E., Saraçlar, M., and Stolcke, A. (2007). Morph-based speech recognition and modeling of out-of-vocabulary words across languages. *ACM Transactions on Speech and Language Processing*, 5(1):3:1–3:29.

Creutz, M. and Lagus, K. (2002). Unsupervised discovery of morphemes. In Maxwell, M., editor, *Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning*, pages 21–30, Philadelphia, PA, USA. Association for Computational Linguistics.

Creutz, M. and Lagus, K. (2004). Induction of a simple morphology for highly-inflecting languages. In *Proceedings of the Seventh Meeting of the ACL Special Interest Group in Computational Phonology*, pages 43–51, Barcelona, Spain. Association for Computational Linguistics.

Creutz, M. and Lagus, K. (2005a). Inducing the morphological lexicon of a natural language from unannotated text. In Honkela, T., Könönen, V., Pöllä, M., and Simula, O., editors, *Proceedings of AKRR'05, International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning*, pages 106–113, Espoo, Finland. Helsinki University of Technology, Laboratory of Computer and Information Science.

Creutz, M. and Lagus, K. (2005b). Unsupervised morpheme segmentation and morphology induction from text corpora using Morfessor 1.0. Technical Report A81, Publications in Computer and Information Science, Helsinki University of Technology.

Creutz, M. and Lagus, K. (2007). Unsupervised models for morpheme segmentation and morphology learning. *ACM Transactions on Speech and Language Processing*, 4(1):3:1–3:34.

Forney, Jr., G. D. (1973). The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278.

Gelas, H., Besacier, L., and Pellegrino, F. (2012). Developments of Swahili resources for an automatic speech recognition system. In *Proceedings of the Third International Workshop on Spoken Languages Technologies for Under-resourced Languages (SLTU'12)*, page 8, Cape Town, South Africa. International Research Center MICA.

Hirsimäki, T., Creutz, M., Siivola, V., Kurimo, M., Virpioja, S., and Pylkkönen, J. (2006). Unlimited vocabulary speech recognition with morph language models applied to Finnish. *Computer Speech & Language*, 20(4):515–541.

Kohonen, O., Virpioja, S., and Lagus, K. (2010a). Semi-supervised learning of concatenative morphology. In *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology*, pages 78–86, Uppsala, Sweden. Association for Computational Linguistics.

Kohonen, O., Virpioja, S., Leppänen, L., and Lagus, K. (2010b). Semi-supervised extensions to Morfessor Baseline. In Kurimo, M., Virpioja, S., and Turunen, V. T., editors, *Proceedings of the Morpho Challenge 2010 Workshop*, pages 30–34, Espoo, Finland. Aalto University School of Science and Technology, Department of Information and Computer Science. Technical Report TKK-ICS-R37.

Kurimo, M., Virpioja, S., and Turunen, V. T. (2010). Overview and results of Morpho Challenge 2010. In *Proceedings of the Morpho Challenge 2010 Workshop*, pages 7–24, Espoo, Finland. Aalto University School of Science and Technology, Department of Information and Computer Science. Technical Report TKK-ICS-R37.

Luong, M.-T., Nakov, P., and Kan, M.-Y. (2010). A hybrid morpheme-word representation for machine translation of morphologically rich languages. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 148–157, Cambridge, MA. Association for Computational Linguistics.

Mermer, C. and Akın, A. A. (2010). Unsupervised search for the optimal segmentation for statistical machine translation. In *Proceedings of the ACL 2010 Student Research Workshop*, pages 31–36, Uppsala, Sweden. Association for Computational Linguistics.

Mihajlik, P., Tüske, Z., Tarján, B., Németh, B., and Fegyó, T. (2010). Improved recognition of spontaneous Hungarian speech — morphological and acoustic modeling techniques for a less resourced task. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(6):1588–1600.

Popović, M. (2011). Morphemes and pos tags for n-gram based evaluation metrics. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 104–107, Edinburgh, Scotland. Association for Computational Linguistics.

Pratt, J. W. (1959). Remarks on zeros and ties in the wilcoxon signed rank procedures. *Journal of the American Statistical Association*, 54(287):655–667.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14:465–471.

Turunen, V. T. and Kurimo, M. (2011). Speech retrieval from unsegmented Finnish audio using statistical morpheme-like units for segmentation, recognition, and retrieval. *ACM Transactions on Speech and Language Processing*, 8(1):1–25.

Virpioja, S. (2012). *Learning Constructions of Natural Language: Statistical Models and Evaluations*. PhD thesis, Aalto University.

Virpioja, S., Kohonen, O., and Lagus, K. (2010). Unsupervised morpheme analysis with Allomorfessor. In *Multilingual Information Access Evaluation I. Text Retrieval Experiments: 10th Workshop of the Cross-Language Evaluation Forum, CLEF 2009, Corfu, Greece, September 30 – October 2, 2009, Revised Selected Papers*, volume 6241 of *Lecture Notes in Computer Science*, pages 609–616. Springer Berlin / Heidelberg.

Virpioja, S., Kohonen, O., and Lagus, K. (2011a). Evaluating the effect of word frequencies in a probabilistic generative model of morphology. In Pedersen, B. S., Nešpore, G., and Skadiņa, I., editors, *Proceedings of the 18th Nordic Conference of*

Computational Linguistics (NODALIDA 2011), volume 11 of *NEALT Proceedings Series*, pages 230–237. Northern European Association for Language Technology, Riga, Latvia.

Virpioja, S., Turunen, V. T., Spiegler, S., Kohonen, O., and Kurimo, M. (2011b). Empirical comparison of evaluation methods for unsupervised learning of morphology. *Traitement Automatique des Langues*, 52(2):45–90.

Virpioja, S., Väyrynen, J. J., Creutz, M., and Sadeniemi, M. (2007). Morphology-aware statistical machine translation based on morphs induced in an unsupervised manner. In *Proceedings of the Machine Translation Summit XI*, pages 491–498, Copenhagen, Denmark.

Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83.

Aalto-ST 25/2013

ISBN 978-952-60-5501-5 (pdf)
ISSN-L 1799-4896
ISSN 1799-4896
ISSN 1799-490X (pdf)

Aalto University
School of Electrical Engineering
Department of Signal Processing and Acoustics
www.aalto.fi

BUSINESS +
ECONOMY

ART +
DESIGN +
ARCHITECTURE

SCIENCE +
TECHNOLOGY

CROSSOVER

DOCTORAL
DISSERTATIONS